Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Jaklič

# Implementacija bitcoina kot plačilni sistem

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Marko Bajec

Ljubljana 2017

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Ko govorimo o plačilnih možnostih na internetu, mislimo predvsem na plačila s kreditnimi karticami in direktnimi plačili preko spletnih bank. V zadnjem času pa so v porastu tudi alternativne plačilne metode, kot so plačevanje z mobilnimi telefoni, s kriptovalutami ali pa kombinacija obojega. V okviru diplomskega dela zasnujte in razvijte sistem, ki bo na osnovi tehnologije bločnih verig prejemal in preverjal plačila, brez da bi se za to posluževal zunanjih storitev. Pri zasnovi sistema izberite najbolj primerne tehnologije in pristope.

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Rok Jaklič sem avtor diplomskega dela z naslovom:

*Implementacija bitcoina kot plačilni sistem*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Marko Bajec,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 18. septembra 2017 Podpis avtorja:

*Zahvaljujem se mentorju dr. Marku Bajcu za vodenje in pomoč pri izdelavi diplomskega dela. Hvala vsem mojim bližnjim za vztrajno podporo na celotni študijski poti. Hvala tudi Urški Telban za končno idejo za temo diplomskega dela.*

Delo posvečam svoji pokojni mami.

# Kazalo

# Seznam uporabljenih kratic

| kratica | angleško | slovensko |
|---|---|---|
| **API** | application programming interface | vmesnik aplikacij |
| **HTTP** | hypertext transfer protocol | protokol za prenos besedil |
| **HTTPS** | hypertext transfer protocol secure | varen protokol za prenos besedil |
| **JSON** | JavaScript Object Notation | JavaScript zapis objektov |
| **ORM** | object-relation mapping | objektno-relacijsko mapiranje |
| **TCP** | transmission control protocol | protokol za nadzor prenosa |
| **WSGI** | web server gateway interface | spletni prehodni vmesnik |
| **ASGI** | asynchronous server gateway interface | asinhroni prehodni vmesnik |
| **SPV** | simple payment verification | preprosto plačilno preverjanje |
| **MITM** | man in the middle attack | napad z vmesnim členom |
| **JSONRPC** | remote procedure call protocol encoded in JSON | protokol oddeljenega klica metode, ki je kodiran z JSON |
| **QR** | quick response code | hitro odzivna koda |

# Povzetek

Ko govorimo o plačilnih možnostih na internetu, mislimo predvsem na plačila s kreditnimi karticami in direktnimi plačili preko spletnih bank. V zadnjem času pa so v porastu tudi alternativne plačilne metode, kot so plačevanje z mobilni telefoni, s kriptovalutami ali pa kombinacija obojega. S porastom uporabe virtualnih valut je, poleg varne hrambe denarja ali premoženja, končnim kupcem in trgovcem omogočena poceni in hitra alternativna metoda izvajanja plačil.

V diplomskem delu razvijemo lastno rešitev prejemanja plačil z bitcoini, ki ni omejena na zunanjo storitev.

**Ključne besede:** bitcoin, plačilni sistem, zaupnost, varnost.

# Abstract

When it comes to payment possibilities on internet, we usually think of payments with credit cards, or paying through online banking. However in recent years because of the raise of cryptocurrencies, alternative payment methods are available such as paying with mobile phone, paying with cryptocurrency or combination of both. With rise of usage of cryptocurrencies, end users and online merchant can choose among alternative payment methods not only as payment method, but also as a store of money and assets.

In this work we implement our own solution for payments with bitcoin, which does not require outside service.

**Keywords:** bitcoin, payment system, trust, security.

# Poglavje 1

# Uvod

S prihodom kriptovalut so se pojavile nove možnosti plačil, ki poleg anonimnosti ponujajo tudi poceni in hiter način plačevanja preko interneta. Spletne trgovine za preverjanje in prejemanje plačil ponavadi uporabljajo zunanje storitve, ki omogočajo preverjanje in prejemanje plačil z več metodami hkrati in se tako izognejo implementacijam različnih sistemov. S prihodom kriptovalut lahko spletne trgovine izvedejo preverjanje in prejemanje plačil s kriptovalutami kar same.

V diplomskem delu želimo postaviti svoj sistem za preverjanje in prejemanje plačil s kriptovaluto bitcoin, ki bo neodvisen od zunanjih storitev ali ponudnikov. Želimo ga zasnovati tako, da bo z nekaj prilagoditvami uporaben tudi za druge trenutno popularne kriptovalute.

## 1.1 Struktura diplomskega dela

Naloga je v nadaljevanju strukturirana na naslednji način:

- v poglavju 2 predstavimo bitcoin in verigo blokov,
- v poglavju 3 predlagamo načrt postavitve sistema,
- v poglavju 4 sledi implementacija načrta,

- v poglavju 5 analiziramo pomanjkljivosti in možne izboljšave implementacije,
- v poglavju 6 zapišemo sklepne ugotovitve.

# Poglavje 2

# Bitcoin in veriga blokov

Trenutni sistemi za plačevanje preko interneta izgledajo na prvi pogled dobro in zadovoljivo, vendar pa imajo plačila z uradno valuto preko interneta svojo ceno. Vse transakcije v uradnih valutah grejo preko banke ali kartičnih podjetij, ki storitev zaračunavajo. Uporabniki večinoma tega ne občutimo neposredno, saj je cena transakcije pogosto že všteta v znesek nakupa, vseeno pa nekateri spletni ponudniki zaradi transparentnosti in ustvarjanja dobrega odnosa s kupci, to zaračunavanje transakcij od bank ali kartičnih podjetij prikažejo tudi na računih. Prav tako, moramo pri izvajanju spletnih nakupov z uradnimi valutami zaupati ne samo ponudniku storitev ali izdelka, ampak še tretji stranki oz. sistemu, ki ponuja plačilne metode. Možnost zlorabe npr. kreditnih kartic, je na vseh nivojih tako izredno velika.

Problem, da je v transakcijo vključena še tretja stranka oz. sistem, so začeli reševati že v osemdesetih letih v prejšnjem stoletju v privatnih podjetjih v ZDA. Dober sistem za anonimno plačevanje je razvil David Chaum[30], ki je s podjetjem DigiCash Inc. bil pionir na področju digitalnih valut. Kljub temu, da so zaradi kriptografskih protokolov transakcije pri tem protokolu, ki jih je razvil David Chaum anonimne, pa je sistem uporabljala samo ena banka. Tudi ta sistem je imel ceno na transakcijo, ki jo je podjetje DigiCash zaračunavalo trgovcem.

Leta 2008 je pod psevdonimom Satoshi Nakamoto na internetu objavil

članek, kjer je bitcoin opisan kot elektronski sistem denarja P2P, ki eliminira potrebo po tretji stranki oz. sistemu baziranem na zaupanju centralnih bank. Namesto tega bitcoin uporablja elektronski sistem plačevanja, ki temelji na kriptografskem dokazu in kjer se vsaka transakcija zapiše v „knjigo“, ki je dostopna vsem.


Slika 2.1: Knjige oz. datoteke bitcoinov na omrežju.

Ustvarjanje in preverjanje transakcij bitcoinov poteka po odprtokodnem kriptografskem protokolu, kjer pri izvedbi transakcije sodelujta neposredno najmanj dve stranki. Problem preverjanja in procesiranja transakcij tudi tukaj ni zastonj, vendar pa je tukaj rešen na zelo eleganten način. „Tistega“, ki preverja in procesira transakcije in je pri tem najbolj uspešen, ga omrežje za narejeno delo nagradi z novimi bitcoini.

Bitcoin je iz vidika navadnega uporabnika digitalna valuta oz. digitalna kriptovaluta, tehnično gledano pa je bitcoin zgolj datoteka bitcoin naslovov

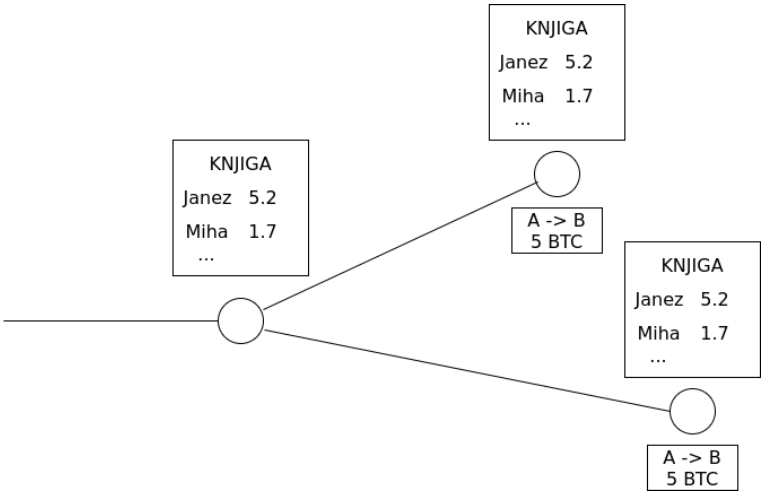
in stanja bitcoinov na določenem bitcoin naslovu.

| | |
|---|---|
| Janez | 5.2 |
| Miha | 1.7 |
| Tina | 0.1222 |
| Mojca | 12.3334 |
| Rok | 11 |
| Valentina | 137 |

Slika 2.2: Knjiga oz. datoteka z bitcoin naslovi in stanji na bitcoin naslovih.

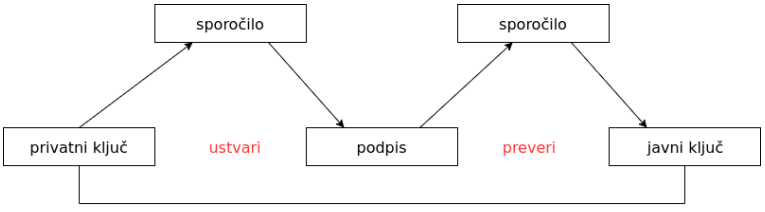
To enako datoteko imajo vsi, ki so udeleženi v omrežju. Prenos bitcoinov na določen naslov se zgodi s spremembo datoteke oz. knjige in to je vidno vsem na omrežju. Posledično vsi na omrežju vidijo vsa stanja na vseh naslovih (naslov si lahko predstavljamo kot bančni račun). Na srečo, so ti naslovi zgolj naključne številke in črke tako, da neka anonimnost že privzeto zagotovljena.

Če želi oseba A poslati neko število bitcoinov osebi B, preprosto „pove“ vsem prisotnim na omrežju, da bo to naredila in vsi prisotni na omrežju posodobijo svojo knjigo z novimi stanji na računih. Da preprečimo goljufanje pri posodabljanju knjig, kjer bi lahko nekdo posodobil knjigo z drugačnimi stanji, bitcoin omrežje zahteva podpis, da je oseba A oz. pošiljatelj res lastnik naslova iz katerega pošilja bitcoine. Podpis temelji na matematičnih oz. kriptografskih funkcijah.




Slika 2.3: Posodabljanje knjige ob pošiljanju bitcoinov.

Ko se ustvari nov bitcoin naslov, pride skupaj s privatnim ključem, ki je matematično povezan s tem naslovom. Denarnica bitcoin vsebuje poljubno število privatnih ključev, ki vsak posebej tvori podpis. Da ustvarimo podpis, se privatni ključ in vsebina transakcije „pošljeta“ v posebno kriptografsko funkcijo. Druga kriptografska funkcija pa omogoča, da lahko s pomočjo nje ta podpis preverimo oz. overimo. Ta overitev nam zagotavlja, da je bil podpis res narejen od prvotnega lastnika in da pripada določeni transakciji. Za razliko od ročnih podpisov, teh podpisov ni mogoče kopirati in ponovno uporabiti v prihodnosti, se pravi so enolični za vsako transakcijo posebej.




Slika 2.4: Ustvarjanje in overitev podpisa.

Čeprav matematični podpisi dokazujejo, kdo je poslal transakcijo, pa ne morejo dokazati kdaj so bile transakcije poslane. To predstavlja problem,

saj lahko zlonamerneˇz pošlje bitcoine dvema osebama „isti trenutek“, kjer je skupni znesek večji, kot ima zlonamrneˇz bitcoinov. Vrstni red je izredno pomemben, saj se s tem določi kateri prejemnik bo prvi plačan. Izkaže se, da je zagotavljanje vrstnega reda pri bitcoinu dokaj zahtevno, saj se bitcoin denarnice, ki tvorijo omrežje nahajajo na različnih delih sveta in prihaja do zakasnitev. Transakcije pridejo do neke denarnice lahko preje, kot do druge in zlonamerneˇz bi lahko o časovnem žigu transakcije lagal. Dva prejemnika bi lahko smatrala, da sta vsak posebej dobila prva transakcijo in tako omogočila zlonamerneˇzu, da zapravi denar dvakrat. Bitcoin taka dejanja preprečuje s tem, da se vsi udeleženi v omrežju odločijo za vrstni red transakcij. Ko se ustvari nova transakcija, gre ta v „bazen“ čakajočih transakcij, od tam pa se razporedijo v ogromno verigo blokov.

Da se odloči katera transakcija gre naslednja v verigo blokov, se igra neka vrste matematična loterija. Udeleženci v loteriji si izberejo čakajočo transakcijo po svoji želji in poskušajo rešiti poseben problem, ki bi v primeru rešitve dodal transakcijo na konec verige blokov. Prva oseba oz. program, ki najde rešitev zmaga in njegova izbrana transakcija se doda kot naslednja v verigo blokov. V resnici izbrano transakcijo tvori več transakcij skupaj. Ta poseben problem, ki je potrebno rešiti se imenuje kriptografska hash funkcija. Funkcija zmeša vhodne parametru in vrne ven znake, ki niso reverzibilni. Program bitcoin hitro vnaša naključne podatke v to funkcijo, ki mora zadoščati določenim kriterijem. Poleg naključnega ugibanja, dodajamo kot v vhod še transakcijo iz bazena čakajočih transakcij in pa izbrano verigo blokov.

Loterija nam tako zagotavlja način, kako vsi udeleženi v omrežju odločijo, katera transakcija je naslednja, ki bo šla na konec verige blokov, prav tako pa matematika zagotavlja, da se vsi s tem strinjajo.

Če se prvič prijavljamo v omrežje, je zaupanja vredna ponavadi tista veriga blokov, ki jo večina uporablja za rudarjenje. Vendar pa bi lahko nekdo hipotetično na svoji verigi blokov „prižgal“ več računalnikov in si tako zagotovil večino. Bitcoin preprečuje to z obveznim reševanje matematičnega problema. Proces določanja vrstnega reda transakcij v verigi blokov vključuje

tudi glasovalni sistem, kjer je del vhoda na koncu verige blokov tudi zmagovalno ugibanje, ki je hkrati tudi glas za verigo blokov. To ima posledico, da ima vsako ugibanje svojo ceno v obliki računanja oz. porabi elektrike, kar bi bilo za zlonamerneža neekonomično, če bi želel večino glasov oz. računske moči uporabiti sebi v prid.

Ker imajo kriptografske hash funkcije definirane statistične lastnosti, lahko izračunamo koliko ugibanj je bilo potrebno za „zmagovalno številko“, nekako tako kot bi želeli oceniti kolikokrat bi morali vreči kovanec, da bi dobili zaporedoma 100x številko. Ko pri ugibanju dobimo zmagovalca, zmagovalec dobi za nagrado na novo narejene bitcoine v svojo bitcoin denarnico. Reševanje teh problemov se imenuje rudarjenje, ki skrbi kako „denar“ prihaja v sistem, glavni namen računanja pa je zagotavljanje, da se vse knjige oz. verige med uporabniki med seboj „strinjajo“. Tak sistem zelo dobro skrbi, kako se denar razprši skozi sistem in kljub temu, da okoli leta 2140 ne bo več na novo narejenih bitcoinov, bodo udeleženci plačani iz pristojbnine nad transakcijami.

Porazdeljeno podatkovno bazo ali datoteko, v katero se podatki zapisujejo v povezavno verigo imenujemo tudi veriga blokov (angl. blockchain). Bitcoin je samo ena izmed možnih implementacij verige blokov. Verige blokov se med seboj razlikujejo predvsem po različnih pravilih, ki morajo biti izpoljnjena, da se lahko na konec neke verige doda nek nov podatek.

## 2.1 Izbira odjemalca bitcoin

Pri izbiri odjemalca bitcoin je v grobem pomembno to, ali se odločimo za takega, ki prenese celotno verigo blokov, v kateri je zgodovina prav vsake izvedene transakcije, ali pa ne. Razlika je v količini podatkov oz. prostora, ki ga bo naš odjemalec zavzel. Če izberemo odjemalec bitcoin, ki prenese celotno verigo blokov, bomo za to potrebovali v času pisanje tega diplomskega dela približno 128 GB prostora.

Kot odjemalec bitcoin ponavadi pojmujemo tudi denarnico bitcoin, ven-

dar ne povsem v vseh primerih.

## 2.2 Polni odjemalec

Odjemalec Bitcoin, ki prenese celotno verigo blokov, lahko preveri, če so veljavni prav vsi prejšnji bloki, kar zagotovlja, da je transakcija veljavna. Primeri polnih odjemalcev so satoshi odjemalec, libbitcoin in btcd.

## 2.3 Lahki odjemalci

Odjemalec Bitcoin, ki prenese iz celotne verige samo glave od vseh blokov v verigi, imenujemo tudi lahek odjemalec. Potreben prostor za hrambo teh podatkov se zmanjša linearno, glede na čas, ko so bile narejene prve transakcije.

Lahki odjemalci se večinoma zanašajo na zunanje strežnike za preverjanje transakcij in izvajajo preprosto plačilno preverjanje (angl. simple payment verification - SPV) na naključnih zunanjih strežnikih in s tem nekako preprečijo napad MITM[25].

Primeri lahkih odjemalcev so bitcoinj, picocoin in pa Electrum.

## 2.4 Zunanje storitve za plačevanje z bitocoini

Kljub temu, da lahko sami postavimo sistem, ki omogoča plačevanje, prejemanje in preverjanje bitcoin plačil, pa obstajajo tudi zunanji ponudniki storitev. Ponudniki se razlikujejo predvsem v ceni storitev in dodatnih funkcijah, kot so npr. direktna pretvorba bitcoinov v želeno valuto po prejetju plačila. Problem zunanjih ponudnikov je, poleg zakasnitve, ki je prisotna ob preverjanju plačila, tudi zaupanje. Namreč, zunanjemu ponudniku moramo zaupati, ali so plačila v kriptovaluti bitcoin prispela ali pa ne, kljub temu, da imamo bitcoine lahko varno shranjene pri sebi. Spomnimo, da lahko plačila preverja vsak, ki ima nameščen poln bitcoin odjemalec.

Zunanji ponudniki storitev za plačevanje z bitcoini so npr. BitPay, CoinGate, BicoinPay, Blockchain, Coinbase, Paxful ...

# Poglavje 3

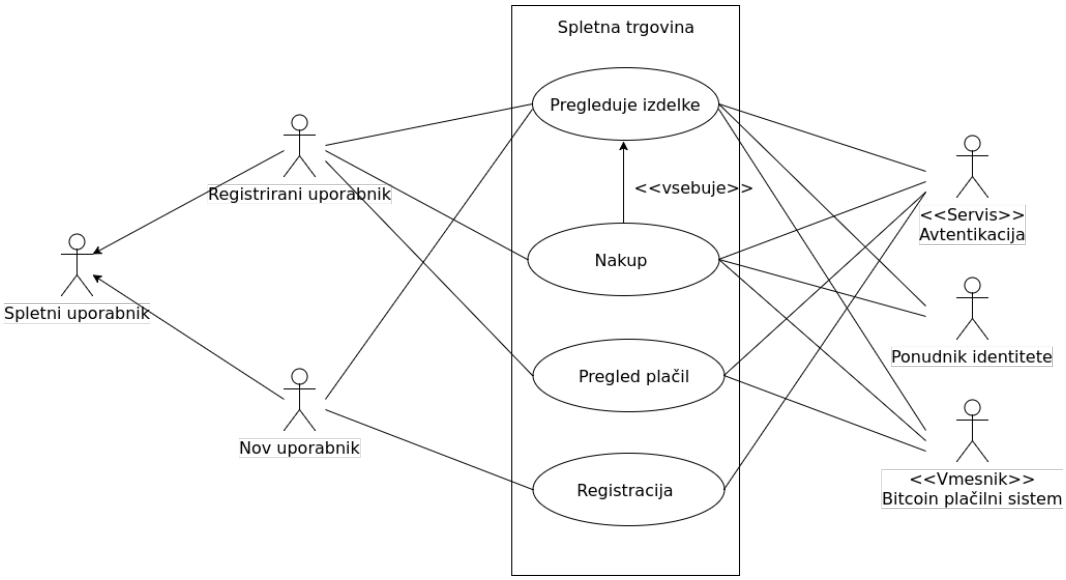
# Načrt

### 3.0.1 Funkcionalosti sistema

Najprej zberemo in analiziramo zahteve uporabnika. Zahteve uporabnika:

- Uporabnik se lahko prijavi.
- Uporabnik lahko pregleduje izdelke.
- Uporabnik lahko izvede plačilo z bitcoin-om za posamezni izdelek.
- Uporabnik lahko pregleduje svoja plačila.

Zahteve uporabnika pretvorimo v funkcionalnosti sistema, katere prikažemo z diagramom poteka. Dodatno ima sistem še dve funkcionalnosti in sicer program za posodabljanje bitcoin cene in program za preverjanje plačil, ki ju opišemo v nadaljevanju.




Slika 3.1: Diagram uporabe spletne trgovine.

## 3.1 Arhitektura sistema

Za potrebe načrta naredimo arhitekturo sistema. Posamezne komponente ločimo na programskem nivoju, saj nam to omogoča večjo fleksibilnost pri sami implementaciji sistema.

Uporabnik se preko protokola HTTP ali WebSocket poveže na spletni strežnik, kateri glede na uporabljen protokol, zahtevke pošlje naprej aplikaciji.

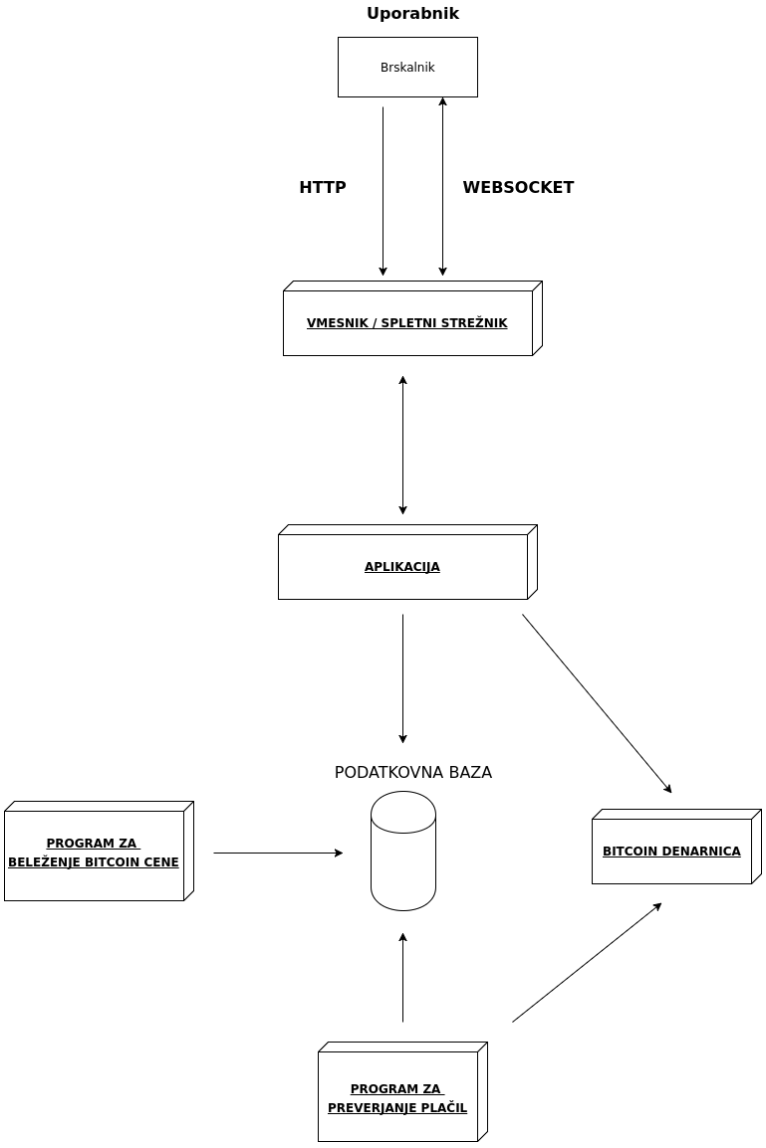
V kolikor pride zahtevek preko protokola HTTP, aplikacija zahtevek obdela v pogledih (angl. views) oz. kontrolerju. Če pride zahtevek preko protokola WebSocket, se zahtevek spremeni v sporočilo, ki se posreduje v dodeljen kanal, kjer delavci v vrsti pobirajo in obdelujejo sporočila. Med obdelavo, strežnik naredi nov kanal, preko katerega vrne odgovor preko vmesnika. Ko delavec obdela sporočilo in naredi svoje delo, lahko po kanalu za odgovor pošlje odgovor, ki je prav tako v obliki sporočila, naš vmesni strežnik

pa ga obdela in pošlje „nazaj“ uporabniku.

Aplikacija uporablja denarnico bitcoin samo za pridobitev novih bitcoin naslovov. Ostale podatke dobi aplikacija preko podatkovne baze, katero pa posodabljata program za pridobivanje bitcoin cene in program za preverjanje plačil.

Program za pridobivanje cene bitcoina pridobiva podatke iz spletne menjalnice Bitstamp, preko WebSocket protokola in preko Pusher servisa. Ceno bitcoina za en evro zapisuje v podatkovno bazo, ki se potem uporablja pri posodabljanju cen v bitcoinih, ko uporabnik pregleduje izdelke.

Program za preverjanje plačil uporablja podatkovno bazo in denarnico bitcoin za preverjanje prispelih plačil v bitcoinih. Ko prispe transakcija na posamezen naslov bitcoin, ki je enoličen za posamezno plačilo, program preveri ali je bilo prispelih dovolj bitcoinov, da aplikacija spremeni status plačila na plačano.




Slika 3.2: Arhitektura sistema.

## 3.2 Načrt podatkovne baze

Za namen tega diplomskega dela bomo v podatkovni bazi hranili podatke o izdelkih, kategorijah, plačilih in podatke o ceni bitcoina. Vendar pa to ne bodo vsi podatki, ki bodo hranjeni v podatkovni bazi. Django privzeto naredi ob prvi migraciji modelov še dodatne tabele za uporabnike, uporabnikovih dovoljenj, skupinah uporabnikov in še nekaj drugih tabel, ki pa za nas niso pomembne in za katere nam ni potrebno narediti logičnih modelov.

Podatki so med seboj povezani s tujimi kluči, eno plačilo se poveže z enim izdelkom, en izdelek pa se poveže z eno kateogrijo.

Našo podatkovno bazo bo iz vidika aplikacije uporabljal samo en uporabnik in sicer uporabnik, ki bo delal poizvedbe za našo spletno stran.

### 3.2.1 Izbira sistema za upravljanje podatkovne baze

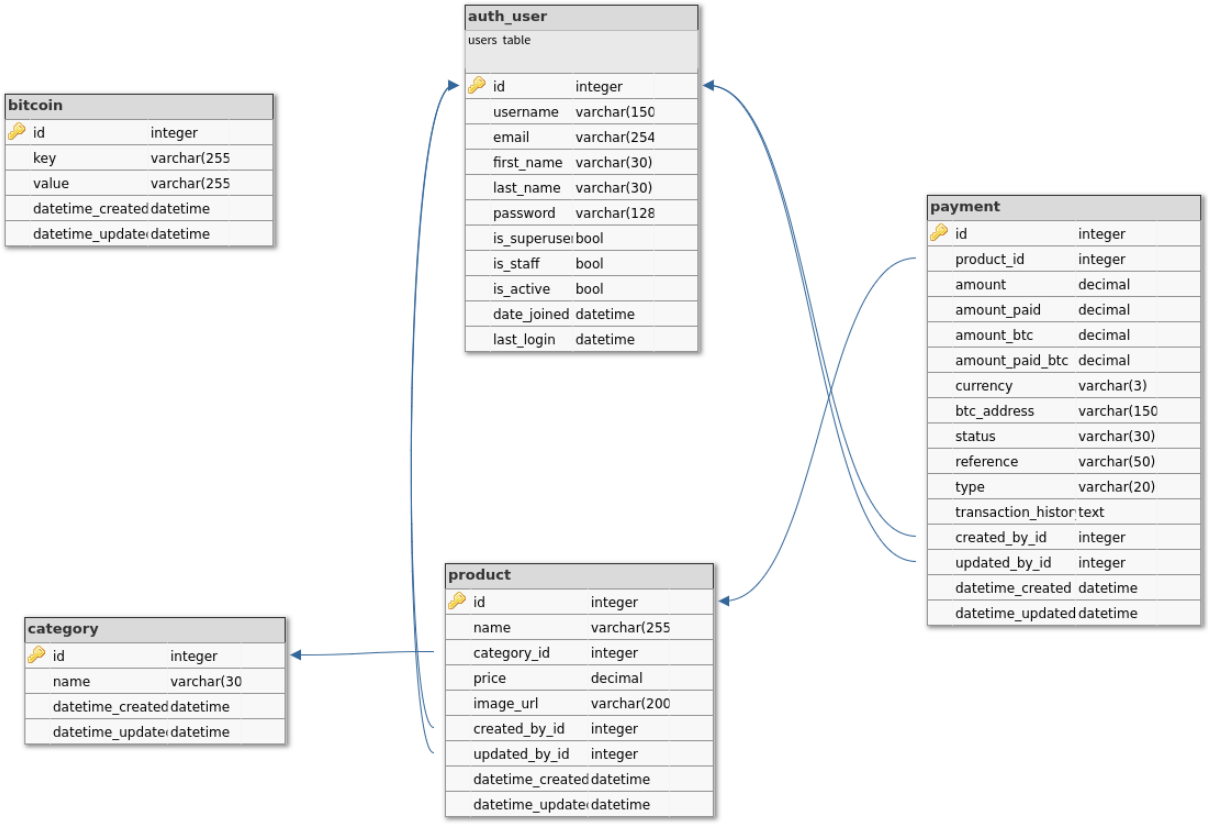
Za sistem za upravljanje podatkovne baze (SUPB) izberemo že preje omenjeno podatkovno bazo SQLite, ki zadostuje našim potrebam, hkrati pa vsebuje velik del lastnosti in funkcij trasakcijskih podatkovnih baz. Tako bomo lahko izdelali relacije, nastavili primarne in tuje ključe in nastavili splošne omejitve.

### 3.2.2 Logični model

Naš logični model sestoji iz petih tabel in sicer:

- tabela auth_user, katera bo vsebovala registrirane uporabnike. Tabela se naredi avtomatsko ob prvi migraciji podatkovne baze in je že prednarejen z definicijo modela v Django ogrodju.
- tabela category, ki bo vsebovala kategorije.
- tabela product, ki bo vsebovala izdelke.
- tabela payment, ki bo vsebovala plačila.

- tabela bitcoin, ki vsebuje dodatne informacije o bitcoin-u, kot je npr. vrednost bitcoina za en evro.




Slika 3.3: Logični model podatkovne baze.

### 3.2.3 Pretvorimo logični model v fizični model

Pretvorbo logičnega modela v fizični model izvedemo med samo implementacijo. Postopek opišemo v poglavju 4.

## 3.3 Načrt postavitve spletnih strani

Za potrebe načrta naredimo preproste skice, katere na koncu uporabimo za izdelavo spletnih strani.

Primarno potrebujemo štiri spletne strani in sicer:

- Spletno stran za prijavo uporabnika.
- Spletno stran z izdelki.
- Spletno stran z izdelki, ki prikazuje QR kodo za nakup izdelka z bitcoini.
- Spletno stran za pregledovanje statusa plačil.


Slika 3.4: Skica spletne strani za prijavo uporabnika.


Slika 3.5: Skica spletne strani z izdelki.


Slika 3.6: Skica spletne strani za pregledovanje statusa plačil.

## 3.4 Načrt implementacije spletnih strani

### 3.4.1 Načrt izdelave predlog

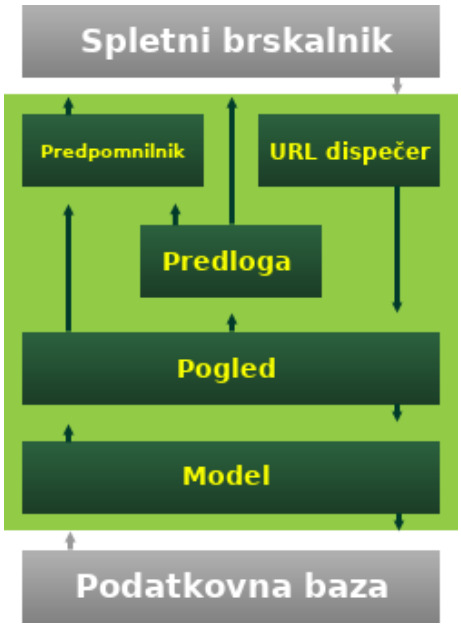
Iz skic, ki smo jih naredili v prejšnjem poglavju naredimo naslednje predloge (angl. templates) v označevalnem jeziku HTML v za ogrodje Django:

- Predloga base.html vsebuje elemente spletne strani, ki so vidni na vseh spletnih straneh naše spletne strani.
- Predloga login.html vsebuje vnosna polja za prijavo na spletno stran.
- Predloga products.html vsebuje izdelke. Na predlogi so tudi potrebni elementi za nakup izdelka.
- Predloga payments.html vsebuje plačila.

Vse naštete predloge so razširjene (angl. extended) iz glavne predloge base.html. To nam prihrani veliko časa in dvojnega dela, saj damo v glavno predlogo tiste stvari in elemente, ki so vidni povsod.

### 3.4.2 Načrt implementacije pogledov

Poglede definiramo za posamezno aplikacijo znotraj projekta v datoteki views.py. Za vsak pogled lahko definiramo v enotni naslov (URL), preko katerega bo pogled prikazal željeno predlogo. V pogledih je ponavadi tudi programska koda za pridobivanje podatkov iz podatkovne baze. Dobljeni podatki se v pogledu obdelajo in pošljejo naprej predlogi, ki se bo prikazala uporabniku. Namesto predloge v HTML obliki, lahko pošljemo nazaj uporabniku tudi podatke v obliki JSON.




Slika 3.7: Zahtevo protokola HTTP v projektih narejenih z Django-m si lahko predstavljamo na prikazan način.

## 3.5 Načrt implementacije zalednih programov

Za naš sistem portebujmo dva zaledna programa. Enega, ki bo zapisoval ceno bitcoina v podatkovno bazo in enega, ki preverjal neplačana plačila za prispela plačila. Zaledna programa bosta tekla v neskončni zanki, zato ji

nastavimo interval po kolikem času naj se željene akcije izvajajo.

## 3.6 Izbira tehnologij in orodij

Za potrebe načrta izberemo operacijski sistem, programe, knjižnice in ogrodja.

### 3.6.1 Linux operacijski sistem

Linux[14] je prost operacijski sistem, podoben Unixu, s prosto dostopno izvorno kodo, zaščiteno s Splošnim dovoljenjem GNU (GNU General Public License - GPL). Kljub monolitični zgradbi jedra je dinamičnost omogočena z nalaganjem oz. odstranjevanjem posameznih modulov. Podpira večprocesorski način, številne datotečne sisteme, možnost izbire razvrščevanj, razširitve pomnilniške enote do 4 GB ali več. Velja za sodobno večopravilno jedro, zasnovano po zamislih iz knjige Operating Systems: Design and Implementation, avtorja Andrewa Stuarta Tannenbauma, profesorja računalništva na Svobodni univerzi v Amsterdamu in tvorca MINIX-a. Uživa podporo (finančno in kadrovsko) podjetij, kot so Sun Microsystems, IBM in Novell.

Izberemo operacijski sistem Linux za razvojni operacijski sistem, čeprav bi lahko izbrali skoraj katerikoli sodobni operacijski sistem, na katerem lahko poženemo Python programsko kodo.

Avtor operacijskega sistema Linux je Linus Torvalds, ki je septembra leta 1991 objavil na spletni novičarski skupini prvo različico operacijskega sistema, ki so ga po njem poimenovali Linux. Takoj so mu priskočili na pomoč številni zanesenjaki z vsega sveta in operacijski sistem je postajal iz dneva v dan zmogljivejši in bolj priljubljen.

### 3.6.2 Python

Python je tolmačitveni programski jezik, ki ga je ustvaril Guido van Rossum leta 1990. Jezik je dobil ime po priljubljeni angleški televizijski nanizanki Leteči cirkus Montyja Pythona (Monthy Python's Flying Circus). Python ima

popolnoma dinamične podatkovne tipe, samodejno upravlja s pomnilnikom in podpira funkcionalno, imperativno oziroma proceduralno, strukturirano in objektno orientirano računalniško programsko paradigmo. Zaradi dinamičnih podatkovnih tipov je podoben jezikom Perl, Ruby, Scheme, Smalltalk in Tcl. Razvili so ga kot odprtokodni projekt, ki ga je upravljala neprofitna organizacija Python Software Foundation.

Programski jezik Python izberemo kot programski jezik za naše zaledne programe, ne samo zaradi razširjenosti in popularnosti med programerji v zadnjem času, ampak tudi zaradi dostopnosti knjižnic, same spletne in tudi akademske podpore.

### 3.6.3 Bitcoin denarnica Electrum

Ker se želimo izogniti pretirani porabi sredstev na računalniku, bomo izbrali bitcoin denarnico Electrum[24], katere namen je hitra, preprosta uporaba denarnice bitcoin, z nizko porabo sistemskih sredstev računalnika. Denarnica Electrum namreč omogoča tehniko SPV[26] preverjanja transakcij. SPV (angl. simple payment verification) oz. preprosto preverjanje plačil je tehnika preverjanja plačil s pomočjo pridobivanja dokazil o vključitvi transakcije v blok transakcij, ki so tam shranjena v obliki drevesa Merkle. Odjemalci SPV morajo tako prenesti samo glave blokov, ki so ponavadi veliko manjše kot celotni bloki v verigi blokov.

Bitcoin denarnica Electrum omogoča dostop do funkcij denarnice preko vmesnika JSONRPC. Na ta vmesnik lahko pošiljamo zahteve HTTP, v katere zapakiramo ukaze v obliki JSON. Dokumentacija do tega vmesnika je prostodostopna in dobro dokumentirana, tako da lahko brez težav vmesnik uporabimo v izbranem zalednem programskem jeziku.

```
1 payload = {
2 "id": "curltext",
3 "method": "getaddressbalance",
4 "params": [btc_address]
5 }
```

Koda 3.1: Primer sporočila, ki ga pošljemo kodirano v JSON obliki

Tu sta pomembna ključa method in params, kjer je funkcija method oz. akcija, ki jo želimo izvesti v denarnici bitcoin, v ključu params pa možni dodatni parametri, ki jih želena akcija morebiti zahteva.

### 3.6.4 SQLite

SQLlite[12] je transakcijska podatkovna baza, ki ni narejena na principu odjemalec-strežnik kot večina novodobnih podatkovnih baz, ampak so podatki shranjeni v datoteki, do katerih dostopamo „direktno“. Ker za svoje delovanje potrebuje le datoteko, je možno tako podatkovno bazo brez težav prenašati med različnimi sistemi, tudi med malimi napravami, kot so predvajalniki MP3, pametni mobilni telefoni in novi televizorji. Podatkovna baza SQLite je na voljo brezplačno, ima vse običajne lastnosti podatkovnih baz SQL in je priljubljena tako v komercialni rabi kot zasebno.

### 3.6.5 Django

Django[16] je brezplačno odprtokodno ogrodje za izdelavo spletnih aplikacij. Napisan je v programskem jeziku Python ter temelji na principu model-pogled-nadzornik (angl. model-view-controller oz. MVC), ki pa ogrodje Django poimenuje malo drugače in sicer model-pogled-predloga (angl. model-view-template oz. MVT).

| MVC | Django (MVT) | se razume kot |
|---|---|---|
| Model | Model | Model/Model |
| Controller | View | Kontroler/Pogled |
| View | Template | Pogled/Predloga |

Slika 3.8: MVC == MVT.

Prednost ogrodja Django je predvsem, da nam omogoča hitro gradnjo spletnih strani in podatkovnih modelov. S pomočjo integriranega ORM-ja prihrani programerjem veliko dela s samo podatkovno bazo oz. stavki SQL, tako da programerjem ostane več časa za samo načrtovanje, oblikovanje in izvedbo spletnih strani. Ob vsaki spremembi modela, Django sam ugotovi, katere spremembe so bile narejene na nivoju modela in naredi migracijsko skripto, ki vsebuje ukaze, ki spremenijo fizični model v podatkovni bazi, da je skladen z našim novim modelom. Django-v ORM sistem omogoča povezave za večino najbolj uporabljanih podatkovnih baz v tem trenutku, vključno s podatkovno bazo SQLite.

Prednost Django ORM-ja je tudi to, da lahko podatkovno bazo brez večjih težav prenesemo na drug sistem za upravljanje podatkovnih baz (SUPB).

### 3.6.6 Django channels

Django channels temelji na posredovanju sporočil/zahtev različnim kanalom WebSocket in obdelovanju/odgovarjanju tem zahtevam. Za to bomo uporabili „Django channels“[18]

Jedro Django channels sestavljajo:

- Kanal (angl. channel), ki je strukturiran v vrsto FIFO. Imamo lahko poljubno število kanalov, odvisno od naših potreb in želja.

- Sporočilo (angl. message), ki je zahteva, ki vsebuje podatke, katere bo lahko „delavec“ obdelal.
- „Delavec“ (angl. consumer) je ponavadi program oz. funkcija, ki obdela sporočilo in sproži nadaljnje korake.
- Vmesni strežnik (angl. interface server) prepozna in zna uporabljati različne protokole. Deluje kot most med Django aplikacijo in zunanjim svetom (npr. spletnim strežnikom).

Da lahko naša spletna stran streže različne protokole (protokol HTTP za samo spletno stran in protokol WebSocket za prejemanje podatkov o plačilih in spremembah cen izdelkov), potrebujemo strežnik ali vmesnik, ki bo zahteve različnih protokolov prepoznal in glede na protokol izbral program, ki bo to zahtevo obdelal ter poslal nazaj podatke, v kolikor je to potrebno.

V našem primeru bo naš vmesnik sprejemal samo zahteve za protokola HTTP in WebSocket. Zelo popularen vmesnik oz. strežnik za ta dva protokola v Pythonu je strežnik Daphne[28]. Ko na vmesnik pride zahteva HTTP ali WebSocket, jo sprejme in spremeni v sporočilo. To sporočilo posredujemo naprej v dodeljen kanal. Že pred tem definiramo kanale za določene zahteve. Za zahteve HTTP pošljemo sporočilo na kanal http.request. Za dohodna sporočila na protokolu WebSocket posredujemo sporočila v kanal WebSocket.receive. Preprosto povedano, ko vmesni strežnik sprejme zahteve (HTTP, WebSocket), jih pretvori v sporočila in jih posreduje/pošlje naprej v kanal za v nadaljno obdelavo.

Ker se kanali lahko napolnijo s sporočili, potrebujemo način, kako bomo ta sporočila obdelovali. Privzeto se sporočila shranjujejo v RAM pomnilnik, vendar se ob večjem številu sporočil zna zgoditi, da vseh sporočil „delavci“ ne bodo zmogli obdelati. Zato je priporočljivo sporočila shraniti v neki bazi, kljub temu pa bomo za naš primer uporabili strežnik Redis[23], ki je preverjena in pogosto uporabljena tehnologija prav za namene začasnega shranjevanja „tekočih“ podatkov v pomnilnik. Torej je potrebno v konfiguraciji CHANNEL_LAYERS nastaviti, da za ključ BACKEND (ki pomeni naš

zaledni strežnik) uporabljamo asgi_redis.RedisChannelLayer.

Tako lahko iz prihajajočih zahtev naši delavci v vrsti pobirajo in obdelujejo poslana sporočila. Medtem, ko se zahteva spreminja v sporočilo, pa naš vmesni strežnik naredi tudi kanal za odgovor, ki ga skupaj s sporočilom posreduje delavcu. Ko delavec obdela sporočilo in naredi svoje delo, lahko po kanalu za odgovor pošlje odgovor, ki je prav tako v obliki sporočila, naš vmesni strežnik pa ga obdela in pošlje „nazaj“ uporabniku.

### 3.6.7 jQuery

jQuery[21] je majhna, hitra in z dodatki bogata JavaScript knjižnica. Zelo dobro se obnese in pomaga pri manipulaciji HTML elementov (dogodki, animacije, AJAX klici ...) in podpira večino trenutno uporabljenih brskalnikov.

Knjižnico je mogoče dobiti brezplačno na internetu.

### 3.6.8 Bootstrap

Bootstrap[22] je odprtokodna knjižnica za načrtovanje in urejanje spletnih strani oz. elementov HTML s pomočjo preddefiniranih razredov CSS oz. stilov. Knjižnica ima že narejene stile za tipografijo, forme, gumbe, navigacijo in ostale uporabniške komponente.

### 3.6.9 Redis

Redis je odprtokodna pomnilniška baza, kjer so podatki shranjeni v pomnilniku na način ključ-vrednost, z možnostjo zadrževanja podatkov. Redis podpira različne abstraktne strukture, kot so nizi znakov, zbirke, seznami ...

### 3.6.10 JSONRPC

JSONRPC[6] je protokol za klice oddaljenih metod, kjer so klici kodirani v obliki JSON. Je zelo preprost protokol in je podoben protokolu XMLRPC, ki ima definiranih le nekaj podatkovnih struktur in ukazov. Protokol JSONRPC

dovoljuje obvestila in večkratne klice na strežnik, ki ne zahtevajo odgovore ali pa odgovori niso v pravilnem vrstnem redu.

### 3.6.11 AJAX

AJAX[7], okrajšava za asinhroni JavaScript in XML[? ], je skupek razvojno spletnih tehnik in tehnologij, ki na uporabnikovi strani uporablja več spletnih tehnologij in ki med drugim omogoča tudi asinhrone klice na spletni strežnik že potem, ko je spletna stran v celoti naložena. Spletne aplikacije lahko tako pošiljajo in prejemajo podatke od/do spletnega strežnika v ozadju, brez vmešavanja v trenutno odprto spletno stran v brskalniku. Z ločevanjem podatkovnega od predstavitvenega nivoja, AJAX omogoča, da lahko dinamično spreminjamo vsebino spletne strani, s čimer se dobi občutek, da je trenutna vsebina na spletni strani tudi realno stanje na strežniku. Do DOM elementov lahko dostopamo dinamično in tako dovolimo uporabniku, da ima z njimi interakcijo.

V praksi se za pošiljanje podatkov od/do strežnika večinoma uporablja struktura JSON namesto strukture XML, ne samo zaradi manjše velikosti podatkov, ki jih je potrebno prenesti za iste opise podatkovnih struktur, ampak tudi zaradi lažje obdelave podatkovnih struktur JSON z JavaScriptom.

### 3.6.12 ECMAScript 6

ECMAScript[8] je standard organizacije Ecma International. Standard je bil narejen z namenom, da se standardizira JavaScript. JavaScript je najbolj znana implementacija standarda ECMAScript. Programerji večinoma uporabljajo ECMAScript za skriptno programiranje na svetovnem spletu, zadnje čase pa je popularen tudi za zaledno programiranje ali programiranje strežniških aplikacij in storitev s pomočjo Node.js.

V grobem prinaša ECMAScript z verzijo 6 podoben način programiranja kot v nižje nivojskih objektno orientiranih programskih jezikih[27], dodatno pa mu ostane še večina lastnosti iz preteklih verzij.

### 3.6.13 WebSocket protokol

Spletni vtični protokol (angl. WebSocket)[9] je računalniški komunikacijski standard oz. protokol, ki podpira dvosmerno komunikacijo čez kanal preko ene povezave TCP.

Spletni vtični protokol je bil prvotno načrtovan za implementacijo v spletnih brskalnikih in spletnih strežnikih, vendar se ga lahko uporabi v katerikoli odjemalski ali strežniški aplikaciji. Edina povezava s protokolom HTTP je, da usklajevanje med odjemalcem in strežnikom, slednji pojmuje kot posodobitveno zahtevo.

Spletni vtični protokol omogoča interakcijo med brskalnikom in spletnim strežnikom z manj začetnega komuniciranja in vzpostavljanja povezave. Ko je povezava enkrat vzpostavljena, ta omogoča dvosmerno komunikacijo in takojšen prenos podatkov, kar daje videz, da se dejanja in akcije izvajajo v realnem času, v kolikor se to uporablja za spletne aplikacije. Po prenosu ostane povezava odprta za nadaljnjo uporabo.

Komunikacija poteka preko vrat TCP številka 80 in 443, kar je priročno predvsem za uporabnike, ki imajo blokirane povezave preko ostalih vrat TCP.

Protokol je trenutno podprt v večini najbolj uporabljenih spletnih brskalnikih, vključno v Google Chrome-u, Microsoft Edge-u, Internet Explorer-ju, Firefox-u, Safari-ju in Operi.

### 3.6.14 WSGI

Spletni prehodni vmesnik (angl. web server gateway interface)[10] je specifikacija za vmesnik med spletnim strežnikom in spletno aplikacijo ali programskim ogrodjem za programski jezik Python[15].

WSGI ima dve strani: strežnik oz. vmesnik (ponavadi spletni strežnik, kot je Apache ali Nginx) in pa aplikacijo oz. aplikativno ogrodje (npr. program ali skripta Python). Ob prihodu zahteve na spletni strežnik, le-ta zažene aplikacijo, poda okoljske nastavitve in nastavi povratno funkcijo aplikaciji. Aplikacija sprocesira zahtevo in vrne odgovor strežniku s pomočjo

povratne funkcije.

### 3.6.15 ASGI

Asinhroni prehodni vmesnik (angl. asynchronous server gateway interface)[11] je specifikacija, ki predlaga standard med strežniškimi protokoli in aplikacijami napisanimi v programskem jeziku Python. Namenjena je za uporabo različnih protokolov, predvsem HTTP, HTTP2 in WebSocket.

Specifikacija WSGI je do sedaj zadostovala potrebam internetnih aplikacij, vendar pa se je s povečevanjem števila uporabnikov in pasovnih širin pokazala možnost hitrejšega in večjega števila zahtev in odgovorov, ki pa ji sčasoma specifikacija WSGI ne bo več zadostovala. Hiter in dinamičen prikaz vsebin je že dandanes postal „tihi“ standard med spletnimi aplikaciji in s specifikacijo ASGI odpravljamo nekaj težav, ki jih specifikacija WSGI ni mogla rešiti.

Glavni cilj specifikacije je, da lahko poleg programske kode za HTTP2 in WebSocket protokola, streže tudi programsko kodo za HTTP protokol in tako zagotavlja interoperabilnost med starimi in novimi spletnimi aplikacijami.

Specifikacija ASGI vsebuje 3 različne komponente:

- Protokolni strežnik (angl. protocol server), ki „prekine“ povezavo na vtičnici in jih prevede v sporočila.
- Aplikacijo, ki živi znotraj strežnika in se ustvari ena na eno vtičnico ob zagonu. Aplikacija obdela sporočila, ki se ustvarijo ob zahtevah.
- Kanali, ki dovoljujejo komunikacijo med različnimi aplikacijami.

Medinstančna komunikacija med objekti je bistvenega pomena. Tako kot pri WSGI-ju, strežnik lahko gosti aplikacijo v sebi in lahko ureja, odpošilja in odgovarja na zahteve. Vendar pa so, za razliko od WSGI-ja, to ustvarjeni objekti, ki za razliko od preprostih klicev funkcij, lahko komunicirajo med sabo, kot točka-do-točke ali kot oddajniki.

## 3.7 Izbira načina implementacije

Novi načini izdelovanja programov in spletnih strani, ki omogočajo dinamičen prikaz spletnih strani s pomočjo definiranih standardov, ki so že podprti v večini spletnih brskalnikov, omogočajo hiter razvoj spletnih aplikacij in prototipov. Vse pogosteje se dogaja, da moramo nekatere uporabniške izkušnje preveriti v „realnem svetu“, z malo časa, ki je namenjen načrtovanju in testiranju, hkrati pa s hitrim odzivnim časom, tako uporabnika kot razvijalca spletnih aplikacij.

Odločimo se, da bomo za različne načine pridobivanja podatkov uporabili:

- za posodabljanje cen izdelkov v bitcoin vrednosti protokol WebSocket,
- za sam nakup izdelka AJAX način pridobivanja podatkov,
- za pridobivanje izdelkov iz podatkovne baze pa način ORM pridobivanja podatkov.

Odločimo se, da bomo za programiranje uporabili naslednje principe:

- objektni način programiranja za uporabniški vmesnik,
- zmes objektnega in proceduralnega programiranja za zaledni del programov.

# Poglavje 4

# Implementacija

Za implementacijo uporabimo v prejšnjem poglavju predlagane tehnologije in principe. Sistem razvijemo v naslednjih korakih:

1. V prvem koraku namestimo strežnika Django, Redis in ostale potrebne pakete za programski jezik Python.

2. V drugem koraku namestimo program Electrum, si ustvarimo denarnico (angl. wallet) ter nastavimo potrebne nastavitve, da program Electrum v ozadju teče kot strežnik, preko katerega dostopamo preko vmesnika JSONRPC.

3. Ustvarimo projekt za ogrodje Django in definiramo podatkovni model.

4. Implementiramo potrebne spletne strani za prijavo uporabnika, nakup izdelkov in pregled plačil.

5. Implementiramo programa za posodabljanje trenutne cene bitcoina v evrih in za pregledovanje neplačanih plačil.

## 4.1 Namestitev strežnika Django, Redis in ostale potrebne pakete za programski jezik Python

```
pacman -S redis-server
pip2 install django
pip2 install asgi_redis
pip2 install channels
systemctl start redis.service
```

Koda 4.1: Ukazi za namestitev strežnika Redis, Django-ta, potrebnih paketov in ukaz za zagon strežnika redis

Nato nastavimo portebno, da Django prepozna strežnik Redis kot zaledni servis, ter napišemo potrebne fukncije na uporabniškem vmesniku za prijavljanja in odjavljanje uporabnikov na/iz kanalov WebSocket:

- nastavimo „Django channels“ v settings.py datoteki, da uporabi asgi_redis kot zaledni sloj za shranjevanje sporočil, poslanih preko protokola WebSocket. Strežnik Redis že pred tem zaženemo z ukazom systemctl start redis.service,
- Dodamo „channels“ v seznam nameščenih programov v spremenljivki INSTALLED_APPS,
- napišemo funkcije, ki bodo dodajale in brisale uporabnike na/iz WebSocket kanalov, na katerih bodo prijavljeni,
- nastavimo „channel_routing“ tako, da kažejo na želene poti v datoteki routing.py. Delavci, ki bodo te kanale uporabljali, bodo vzpostavili svoj „zasebni kanal“ med strežnikom in delavcem.

```

@channel_session_user_from_http
def ws_connect(message):
  Group("payments-%s" % message.user.username).add(message.
      reply_channel)
  message.reply_channel.send({"accept": True})

@channel_session_user_from_http
```

```
8  def ws_disconnect(message):
9    Group("payments-%s" % message.user.username).discard(
         message.reply_channel)
10   message.reply_channel.send({"close": True})
```

Koda 4.2: Programska koda za prijavo in odjavo uporabnika na/s WebSocket kanalov

## 4.2 Namestitev programa Electrum

### 4.2.1 Electrum namestitev

Electrum program je narejen v programskem jeziku Python, zato lahko namestimo programski paket z naslednjim ukazom v lupini (pod administrativnim uporabniškim imenom):

```
1  pip2 install https://download.electrum.org/2.9.2/Electrum
       -2.9.2.tar.gz
```

Koda 4.3: Namestitev Electrum program s pip2 namestitvenim programom

### 4.2.2 Ustvarimo denarnico in zaženemo Electrum program

```
1 electrum create
2 electrum daemon start
```

Koda 4.4: Zagon Electrum programa v ozadju

### 4.2.3 Omogočimo vmesnik JSONRPC na programu Electrum

```
1 electrum setconfig rpcport 7777
```

## 4.3 Definicija podatkovnih modelov

Iz načrta predstavljenega v poglavju 3, ustvarimo podatkovne modele za naslednje tabele v podatkovni bazi:

- tabela bitcoin ima v Django ORM-ju definiran model Bitcoin
- tabela category ima v Django ORM-ju definiran model Category
- tabela product ima v Django ORM-ju definiran model Product
- tabela payment ima v Django ORM-ju definiran model Payment

V modelu Product definiramo najbolj osnovne atribute za predstavitev nekega izdelka. Atributa, ki ju nujno potrebujemo, sta ime (angl. name) in cena (angl. price) izdelka. Zaradi lažje narave izdelave podatkovnega modela, v modelu Product ne bomo pretvarjali cen iz evra v bitcoine, ampak bomo pretvorbo delali pri samem plačilu izdelka. Prav tako se bomo omejili samo na nakup enega izdelka z enim plačilom, saj želimo v diplomski nalogi primarno predstaviti implementacijo svojega plačilnega sistema za plačevanje v kriptovaluti bitcoin.

V modelu Payment definiramo atribute, ki so potrebni za predstavitev plačila. Atributi, ki jih nujno potrebujemo, so znesek (angl. amount), znesek v bitcoinih (angl. amount_btc), znesek prispelih bitcoinov na naslov (angl. amount_paid_btc), naslov bitcoin (angl. btc_address) in pa status.

Kljub temu, da nam večina orodij za načrtovanje podatkovnih baz sama pretvorijo logične modele v stavke SQL za željeno podatkovno bazo, pa naprednejša ogrodja za izdelavi spletnih strani in aplikacij kot so Django ali Ruby on Rails[17], omogočajo izdelavo logičnih in programskih modelov za podatkovno bazo v enem koraku.

Posamezne modele definiramo na nivoju aplikacije, podatkovno bazo pa uporabljamo v željenem programskem jeziku s pomočjo ORM-ja od Djanga-ta.

```
1  # -*- coding:utf-8 -*-
2  import datetime
3  from django.db import models
4  from django.utils.translation import ugettext_lazy as _
5  from common.models import Skeleton, SkeletonU
6
7  class Category(Skeleton):
8    name = models.CharField(max_length=30, blank=False, null=
       False)
9
10   def __unicode__(self):
11     return self.name
12
13   class Meta:
14     db_table = 'category'
15
16 class Product(SkeletonU):
17   name = models.CharField(max_length=255, blank=False, null=
       False)
18   category = models.ForeignKey(Category)
19   price = models.DecimalField(max_digits=12, decimal_places
       =2, blank=True, null=True)
20   image_url = models.URLField(blank=True, null=True)
```

```
21
22    def __unicode__(self):
23        return "%s - %s" % (self.category.name, self.name)
24
25    class Meta:
26        db_table = 'product'
```

Koda 4.5: Definicija modelov Category in Product, ki predstavljata v podatkovni bazi tabeli category in pa product

Ko definiramo modele s pomočjo ORM-ja, je potrebno zgolj narediti migracijo na nivoju Django-ta. To izvedemo z naslednjima dvema ukazoma:

```
1
2 python2 manage.py makemigrations # pregleda trenutne modele
      za spremembe na nivoju podatkovne baze. Ce je bila kaksna
      sprememba, Django to zazna in naredi migracijsko skripto.
3 python2 manage.py migrate # narejene sprememe oz. migracijsko
       skripto lahko potem pozenemo s to komando.
```

Koda 4.6: Migracija podatkovnih modelov s pomočjo Django ORM-ja

Dostop do željenih podatkov v podatkovni bazi izvajamo potem preko objektov posameznih definiranih modelov.

Del ORM-ja za ogrodje Django sam sestavi potrebne stavke SQL za pridobitev podatkov iz podatkovne baze. Dobljeni podatki so že pretvorjeni v programske objekte definiranega modela in lahko izvajamo manipulacijo nad željenim objektom preko programskega jezika namesto s stavkom SQL, kar ima svoje prednosti, pa tudi slabosti. Prednosti so predvsem v lažjem in bolj prijaznem dostopu do željenih podatkov na nivoju aplikacije, kar pomeni tudi lažje delo programerju. Sodobna ogrodja ORM omogočajo tudi izvajanje zahtevnejših poizvedb, npr. z večimi povezavami med tabelami in agregaciji podatkov. Kljub temu, da lahko pridobivamo podatke na nivoju aplikacije preko ORM-ja, pa nam Django ogrodje dovoljuje izvajanje stavka SQL direktno na podatkovno bazo. Na koncu je odvisno od programerja, kdaj bo izbral željen princip za pridobitev podatkov iz podatkovne baze, kar je največ kar si lahko želimo, saj imamo svobodo izbire.

## 4.4 Implementacija spletnih strani in pogledov

Za namen diplomskega dela implementiramo spletne strani, za katere smo naredili načrt v poglavju 3:

1. za prijavo uporabnika (vhodna spletna stran),

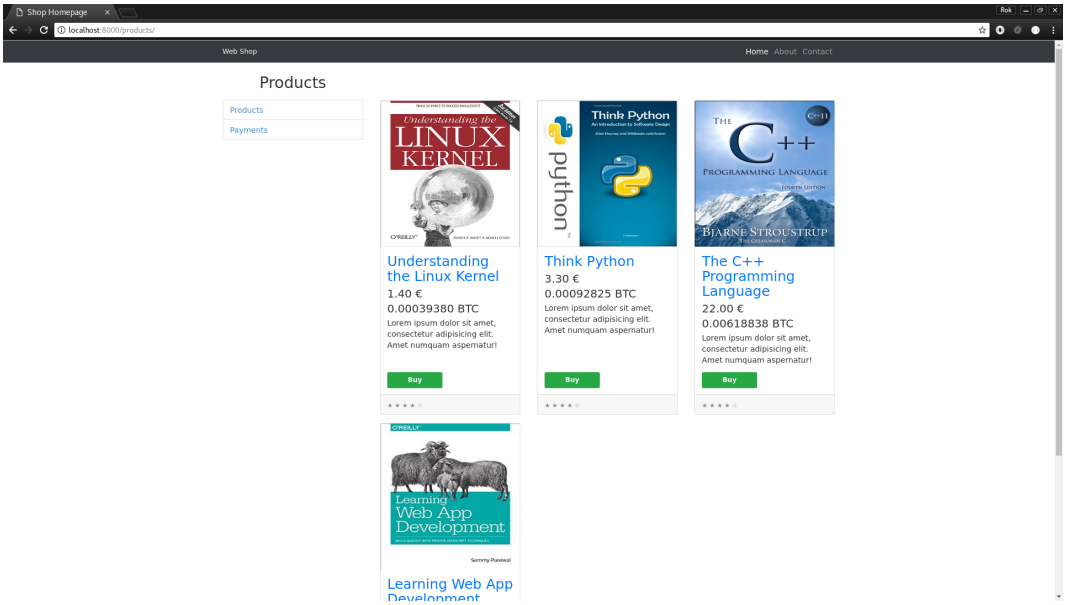
2. za pregled in nakup izdelkov,

3. za pregledovanje statusa plačil.

### 4.4.1 Spletna stran za prijavo uporabnika (vhodna spletna stran)

Če nismo prijavljeni, nam vhodna spletna stran prikaže vnosni polji za uporabniško ime in geslo. V kolikor smo že prijavljeni ali pa po uspešni prijavi, nas spletna stran avtomatsko preusmeri na stran z izdelki.

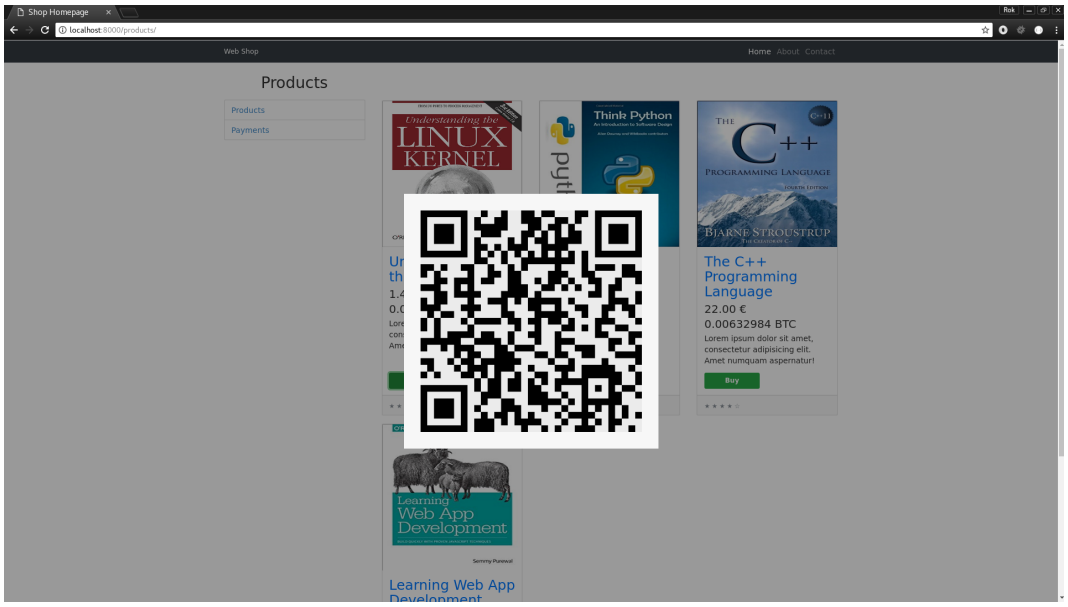
### 4.4.2 Spletna stran z izdelki




Slika 4.1: Spletna stran z izdelki.

Spletna stran z izdelki prikazuje, poleg cene v evrih, tudi ceno v bitcoinih, ki pa se dinamično spreminja v skoraj relanem času, in sicer glede na trenutni tečaj bitcoina v evrih. Spletna stran oz. uporabniški vmesnik smo v tem primeru sprogramirali tako, da preverja in posodablja cene prikazanih izdelkov v bitcoinih na vsaki dve sekundi. Tako se preko protokola WebSocket na strežnik vsaki dve sekundi pošlje zahteva oz. sporočilo, da želimo dobiti posodobljene cene izdelkov.

Po izbiri produkta in s klikom na gumb kupi (angl. buy) se nam prikaže koda QR, ki predstavlja bitcoin URI[29], v katerem je zapakiran bitcoin naslov in znesek, ki ga moramo plačati.




Slika 4.2: Koda QR po kliku na gumb za nakup izdelka, v katerem je zakodiran bitcoin naslov in znesek za plačilo.

Da pridobimo bitcoin URI za izbran izdelek, potrebujemo znesek za plačilo in enoličen bitcoin naslov.

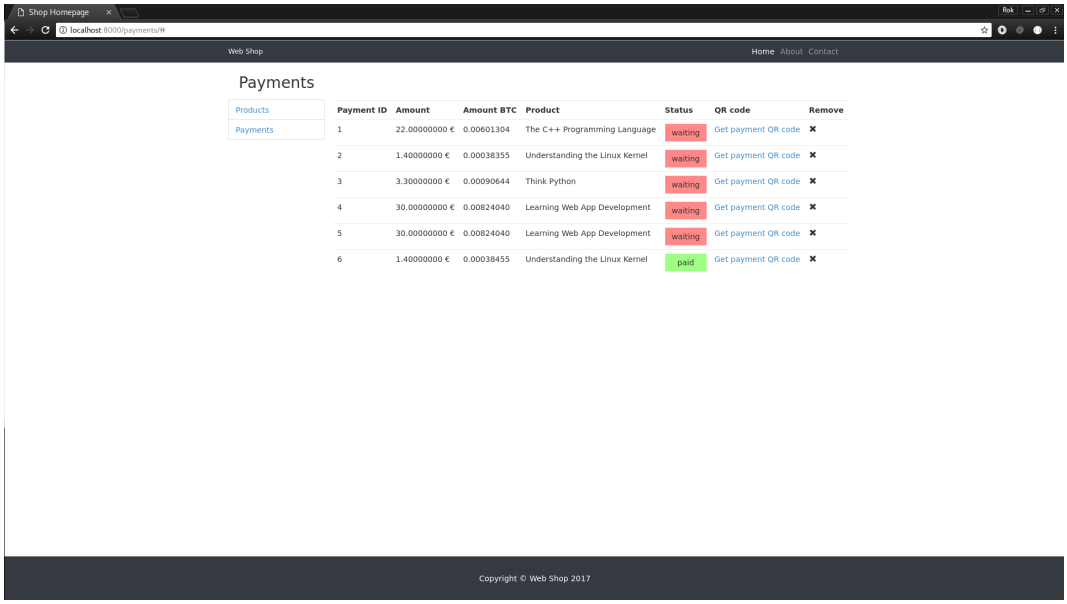
Za objekt tipa Payment, nam v zaledju sistema izračuna funkcija calculate_btc_price, ki nam, iz prej pridobljene vrednosti bitcoina v evrih, izračuna vrednost izdelka v evrih. Ceno vrednosti bitcoina v evrih dobimo s programom price_watcher.py, ki je opisan v nadaljevanju.

Ko pridobimo ceno izdelka, lahko naredimo zahtevo oz. klic JSONRPC na našo bitcoin denarnico z metodo „addrequest“. Za klic te metode moramo podati v atribut „params“ dva parametra v obliki asociativnega seznama, v katerem je en atribut „force“ nastavljen na „true“, drugi atribut „amount“ pa ima vrednost zneska, ki ga želimo dobiti na želeni bitcoin naslov. S tem klicem pridobimo iz naše bitcoin denarnice še neuporabljen, enoličen bitcoin naslov.

```
def add_request(amount):
  try:
  payload = {
    "id": "curltext",
    "method": "addrequest",
    "params": {"force": True, "amount": amount}
  }

  headers = {'Content-type': 'application/json'}

  request = requests.post(BITCOIN_RPC_ADDRESS, json=payload,
      headers=headers)

  if request.status_code != 200:
    print("could not connect to bitcoin rpc")

    return ""

  return request.json()['result']

  except Exception as error:
    subject = "ERROR when getting transactions from bitcoin
        client or something..."
    message = "see subject"

    print(error)
```

Koda 4.7: Programska koda za pridobitev enoličnega bitcoin naslova iz Electrum bitcoin denarnice

### 4.4.3 Spletna stran za pregledovanje statusa plačil




Slika 4.3: Spletna stran za pregledovanje status plačil.

Spletna stran za pregledovanje plačil nam prikazuje statuse plačil. V uporabniškem vmesniku v razredu Payment (JavaScript), atributu this.payments_socket, v katerem je objekt tipa WebSocket, nastavimo atribut onmessage na željeno funkcijo, ki se bo izvedla, ko bo prispelo sporočilo iz spletnega strežnika. Ker smo sami definirali poslano sporočilo, nam tega ne bo težko razbrati iz oblike JSON v brskalniku. Funkcija se bo izvedla samo, če bo prišlo sporočilo iz strežnika preko povezave WebSocket.

```
1 payments_socket.onmessage = function(message) {
2   var data = JSON.parse(message.data);
3
4   $(".payment-status[data-payment-id='" + data["id"] + "']").
        attr("data-payment-status", data["status"]);
5
6   $(".payment-status[data-payment-id='" + data["id"] + "']").
        text(data["status"]);
7 };
```

Koda 4.8: Programska koda oz. povratni klic funkcije, ki se izvede po prispelem sporočilu iz WebSocket kanala, na katerem je povezan uporabnik

## 4.5 Implementacija zalednih programov

### 4.5.1 Program za beleženje bitcoin cene v evrih

Program za beleženje bitcoin cene v evrih se bo imenoval price_watcher.py, ki bo tekel v ozadju našega sistema. Program preverja ceno bitcoina na spletni menjalnici Bitstamp in ceno uporabi za pretvorbo vrednosti enega evra v število bitcoinov. To število potem program zapiše v podatkovno bazo. Cena bitcoina v evrih se uporablja za pretvorbo cen, npr. izdelkov v bitcoin vrednosti.

Uporabili bomo storitev (angl. service) Pusher od spletne menjalnice Bitstamp. Program se poveže preko protokola WebSocket na storitev Pusher, preko katerega se pridobiva trenutno vrednost bitcoina v evrih.

V vsaki storitvi Pusher imamo dodeljen enoličen ključ, ki identificira „ponudnika“, na katerega se povezujemo. V našem primeru se bomo s tem ključem povezali na ponudnika Bitstamp in na kanal „live_trades_btceur“, ki nam ob vsakem dogodku „trade“, ki pomeni nakup oz. prodajo bitcoina v evrih na spletni menjalnici Bitstamp, pošlje sporočilo, da je bila opravljena transakcija. Tako dobimo ceno bitcoina v skoraj realnem času, katero lahko

potem naprej uporabimo za pretvorbo pretvorimo vrednost bitocoinov za en evro.

```
1
2  def callback(data):
3    price = json.loads(data)['price']
4    datetime_updated = datetime.datetime.utcnow().replace(
         tzinfo=pytz.UTC)
5
6    update_btc_price("bitstamp_1_eurbtc", price,
         datetime_updated)
7
8  def connect_handler(data):
9    channel = pusher.subscribe('live_trades_btceur')
10   channel.bind('trade', callback)
11
12 appkey = "de504dc5763aeef9ff52"
13 pusher = pusherclient.Pusher(appkey)
14 pusher.connection.bind('pusher:connection_established',
       connect_handler)
15 pusher.connect()
16
17 while True:
18   time.sleep(1)
```

Koda 4.9: Programska koda, ki poskrbi, da se ob dogodku "trade"povratno pokliče funkcijo, ki poskrbi za posodobitev cene bitcoina v evrih v naši podatkovni bazi.

### 4.5.2 Program za preverjanje prejetih plačil

Program za preverjanje prejetih plačil oz. transakcij se imenuje payment_watcher.py. Program preverja vsa neplačana plačila ali so bila v celoti plačana.

Tehnično gledano, program preko modela Payment preverja vsa plačila, ki imajo nastavljena atribut status na „WAITING“, ali je bila izvedena kakšna transakcija na bitcoin naslovu za navedeno plačilo v naši bitcoin denarnici.

Če je bila na bitcoin naslovu izvedena kakšna transakcija in če je skupen znesek iz transakcij enak ali večji od želenega zneska plačila, potem se plačilu nastavi atribut „status“ na „PAID“. Hkrati pa se, v kolikor je nek spletni uporabnik prijavljen na kanal WebScoket payments-%s (kjer je %s enolična številka uporabnika), ob posodobitvi statusa na „PAID“ pošlje sporočilo, da je plačilo prispelo.

```
1  def get_balance(btc_address):
2  try:
3    payload = {
4      "id": "curltext",
5      "method": "getaddressbalance",
6      "params": [btc_address]
7    }
8
9    headers = {'Content-type': 'application/json'}
10   request = requests.post('http://127.0.0.1:7777', json=
         payload, headers=headers)
11
12   if request.status_code != 200:
13     print("could not connect to bitcoin rpc")
14
15   return request.json()['result']
16
17 except Exception as error:
18   print(error)
```

Koda 4.10: Programska koda, ki pridobi stanje bitcoinov na bitcoin naslovu preko klica JSONRPC na bitcoin denarnico.

```
while True:
  payments = Payment.objects.      (status=WAITING)
  for p in payments:
    if p.btc_address is not None:
      balance = get_balance(p.btc_address)

      if Decimal(balance['confirmed']) >= p.amount_btc:
        p.amount_paid_btc = Decimal(balance['confirmed'])
        p.status = PAID
        p.save()
      elif Decimal(balance['unconfirmed']) >= p.amount_btc:
        p.amount_paid_btc = Decimal(balance['unconfirmed'])
        p.status = PAID
        p.save()

      send_payment(p)

  time.sleep(2)
```

Koda 4.11: Programska koda, ki spremlja vsa čakajoča plačila.

Uporabnik dobi informacijo o tem, ali je uspešno nakazal bitcoine na zahtevani naslov, preko spletnega vtičnika.

```
def send_payment(payment):
  data = {
    "id": payment.  ,
    "status": "paid"
  }

  Group("payments-%s" % payment.created_by).send({'text':
     json.dumps(data)})
```

Koda 4.12: Programska koda, ki poskrbi za pošiljanje obvestila na vzpostavljeni kanal med uporabnikom in delavcem, preko spletnega vtičnika.

# Poglavje 5

# Analiza implementacije sistema

## 5.1 Varnost

Pri postavitvi plačilnih sistemov je v ospredju vedno vprašanje varnosti. Izhajati moramo namreč iz dejstva, da plačilni sistemi, ki prejemajo in preverjajo plačila, niso nikoli dovolj varni.

Nenapisano pravilo pri postavitvi takih sistemov je, da dovoljujemo in dodajamo samo toliko dostopov do sistemov in podsistemov, kolikor je potrebno in nič več. Če je možno, sisteme čim bolj razčlenimo. Že s temi preprostimi pravili dosežemo to, da v kolikor napadalec zlorabi en podsistem, mora vložiti isti trud ali še več, da bo prišel na ostale nivoje, kar pa nam daje večjo verjetnost, da nepravilnosti pri delovanju plačilnih sistemov hitreje opazimo.

Prav tako je nenapisano pravilno, da sisteme, podsisteme, povezave med njimi in podatke šifriramo kjer je le mogoče. Će je mogoče, šifriramo naslednje nivoje:

- Šifriramo najprej razdelek na trdem disku (angl. disk partition), kamor bomo naložili operacijski sistem in podatke.
- Šifriramo ključne podatke v podatkovnih bazah, kjer je le mogoče na nivoju aplikacije.

- Šifriramo povezave med brskalnikom in spletno stranjo (HTTPS).
- Šifriramo povezave do podatkovne baze in povezave do bitcoin denarnice.
- Dostope omejimo uporabnikom do podatkovne baze samo za tiste stvari, ki so potrebne.

Tako smo med implementacijo opazili, da bi poleg naštetega, podisteme lahko zavarovali tako, da bi jih čim bolj razčlenili. Tako bi spletni strežnik oz. vmesnik, aplikacijo, podatkovno bazo in bitcoin denarnico namestili s predlaganimi varnostnimi postopki na svoje sisteme. Tako bi se izognili varnostnim pomanjkljivostim v primeru vdora na samo spletno stran. Spletna stran bi tako imela samo pravico vpogleda v bitcoin denarnico, ne pa tudi možnosti izvajanja ukazov za pošiljanje bitcoinov na nek tretji naslov.

## 5.2 Pomanjklivosti

### 5.2.1 Cena bitcoina

Kljub temu, da je bitcoin kot valuta prisotna že več kot 8 let, je možno dnevno nihanje vrednosti valute tudi za več 10% vrednosti. Da bi se izognili izgubi vrednosti bitcoina in s tem tudi denarja, bi lahko v trenutku, ko bi prejeli plačilo v obliki bitcoinov, te iste bitcoine prenesli na neko spletno menjalnico in jih takoj prodali. Tako bi se izognili možnim velikim nihanjem vrednosti bitcoina. Hkrati se poraja ideja, da bi mogoče bitcoine varčevali in si s tem večali vrednost portfelja, kar pa bi imelo pozitiven učinek na naše poslovanje. Tak princip uporablja spletna trgovina Overstock, ki 50% vrednosti nakupa v bitcoinih obdrži za nadalnje investicije[31].

### 5.2.2 Podatkovna baza

Hipotetično, če bi bila naša spletna stran nameščena na nekem spletnem strežniku in bi imela veliko uporabnikov, bi bilo na naši podatkovni bazi

največ poizvedovanja po plačilih in sicer po tistih, ki še niso bila plačana. Poizvedovanja po neplačanih plačilih se izvajajo vsako sekundo, saj hočemo plačilo nastaviti na status kot plačano v skoraj realnem času. V kolikor bi bilo veliko takih plačil, bi morali verjetno izbrati drugačen pristop in sicer bi morali implementirati našo bitcoin denarnico tako, da bi se na dogodek, ko se je zgodila kakšna transakcija na nekem naslov v naši bitcoin denarnici, preverili in izvedli željeno akcijo nad plačilom, ki se ujema z naslovom te transakcije. Prav tako bi na tej spletni strani bilo veliko poizvedovanja po izdelkih in njihovih cenah, za kar bi bilo smiselno narediti mehanizem pogleda (angl. view) nad temi podatki.

Če na hitro pomislimo, bi lahko posodabljanje bitcoin vrednosti izdelkov izvedli s prožilci na sami podatkovni bazi vsakič, ko se posodobi vrednost bitcoina za en evro. Vendar pa bi bilo tako posodabljanje ob veliki količini izdelkov preobremenjujoče za našo podatkovno bazo, zato je posodabljanje bitcoin vrednosti izdelka takrat, ko uporabnik pogleda ceno za določen izdelek, najbolj smiselno.

# Poglavje 6

# Sklepne ugotovitve

V tem diplomskem delu smo pokazali, da je za postavitev svojega bitcoin plačilnega sistema ključno predvsem široko poznavanje sodobnih programskih tehnologij in principov, arhitekturnih zasnov, ki jih lahko med seboj brez večjih težav povezujemo in spreminjamo. Poleg poznavanja programskih tehnologij je za postavitev sistema potrebno tudi poznavanje varnostnih principov in metod.

Arhitekturno gledano, lahko predlagana rešitev v diplomskem delu z nekaj izboljšavami in razširitvami brez večjih težav pokrije plačila z bitcoini za celotno Slovenijo, če predvidevamo, da je v Sloveniji izdanih približno 80-100 računov na sekundo v obdobju, ko je izdajanje računov najbolj prisotno.

# Literatura


[1] Bitcoin. [Online]. Dosegljivo: http://en.wikipedia.org/wiki/Bitcoin. [Dostopano 11. 8. 2017].

[2] Blockchain. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Blockchain. [Dostopano 11. 8. 2017].

[3] SQL. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/SQL. [Dostopano 14. 8. 2017].

[4] ORM. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Object-relational_mapping. [Dostopano 14. 8. 2017].

[5] JSON. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/JSON. [Dostopano 14. 8. 2017].

[6] JSONRPC. [Online]. Dosegljivo: http://www.jsonrpc.org/specification. [Dostopano 14. 8. 2017].

[7] AJAX. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Ajax_(programming). [Dostopano 14. 8. 2017].

[8] ECMAScript 6. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/ECMAScript. [Dostopano 14. 8. 2017].

[9] WebSocket protokol. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/WebSocket. [Dostopano 14. 8. 2017].

[10] WSGI. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface. [Dostopano 14. 8. 2017].

[11] ASGI. [Online]. Dosegljivo: http://channels.readthedocs.io/en/stable/asgi.html. [Dostopano 14. 8. 2017].

[12] SQLite. [Online]. Dosegljivo: https://www.sqlite.org/about.html. [Dostopano 11. 8. 2017].

[13] XML. [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/XML. [Dostopano 11. 8. 2017].

[14] Linux operating system. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Linux. [Dostopano 11. 8. 2017].

[15] Python programming language. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Python_(programming_language). [Dostopano 11. 8. 2017].

[16] Django web development framework. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Django_(web_framework). [Dostopano 11. 8. 2017].

[17] Ruby on Rails web development framework. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Ruby_on_Rails. [Dostopano 11. 8. 2017].

[18] Django channels. [Online]. Dosegljivo: https://channels.readthedocs.io/en/stable/concepts.html. [Dostopano 11. 8. 2017].

[19] Arch Linux installation guide. [Online]. Dosegljivo: `https://wiki.archlinux.org/index.php/installation_guide`. [Dostopano 11. 8. 2017].

[20] Arch Linux distribution. [Online]. Dosegljivo: `https://wiki.archlinux.org/index.php/Arch_Linux`. [Dostopano 11. 8. 2017].

[21] jQuery. [Online]. Dosegljivo: `https://en.wikipedia.org/wiki/JQuery`. [Dostopano 11. 8. 2017].

[22] Bootstrap. [Online]. Dosegljivo: `https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)`. [Dostopano 11. 8. 2017].

[23] Redis. [Online]. Dosegljivo: `https://redis.io/`. [Dostopano 11. 8. 2017].

[24] Electrum Bitcoin client. [Online]. Dosegljivo: `https://en.bitcoin.it/wiki/Electrum`. [Dostopano 11. 8. 2017].

[25] Man in the middle attack. [Online]. Dosegljivo: `https://en.wikipedia.org/wiki/Man-in-the-middle_attack`. [Dostopano 11. 8. 2017].

[26] Simple payment verification. [Online]. Dosegljivo: `https://bitcoin.org/en/glossary/simplified-payment-verification`. [Dostopano 11. 8. 2017].

[27] Object oriented programming. [Online]. Dosegljivo: `https://en.wikipedia.org/wiki/Object-oriented_programming`. [Dostopano 11. 8. 2017].

[28] Daphne, a HTTP, HTTP2 and WebSocket protocol server. [Online]. Dosegljivo: `https://github.com/django/daphne`. [Dostopano 11. 8. 2017].

[29] URI [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier. [Dostopano 7. 8. 2017].

[30] David Chaum [Online]. Dosegljivo: https://en.wikipedia.org/wiki/David_Chaum. [Dostopano 7. 8. 2017].

[31] OSTK to HODL: Overstock to Keep 50% of All Bitcoin Payments as Investments [Online]. Dosegljivo: https://www.coindesk.com/ostk-hodl-overstock-keep-50-bitcoin-payments-investmen [Dostopano 7. 8. 2017].